Une méthode de débogage d'ontologies OWL basées sur la détection d'anti-patrons

Catherine Roussey, François Scharffe, Oscar Corcho, Ondrej Zamazal

Une méthode de débogage d'ontologies OWL basées sur la détection d'anti-patrons.

Catherine Roussey[1, 2], Francois Scharffe[3], Oscar Corcho[4], Ondrej Zamazal [5]

[1] Cemagref, Campus des Cézeaux, Clermont Ferrand, 24 avenue des Landais, BP 50085, 63172 Aubière cedex, France
[2] Université de Lyon CNRS, Université Lyon 1, LIRIS UMR5205, Villeurbanne, France
[3] INRIA Grenoble Rhône-Alpes, 655 avenue de l'Europe, Montbonnot Saint Martin, 38334 Saint-Ismier, France
[4] Ontology Engineering Group. Departamento de Inteligencia Artificial. Universidad Politécnica de Madrid, Spain
[5]UEP, Prague, Knowledge Engineering Group, Czech Republic

Résumé : Le débogage d'ontologies OWL incohérentes est une tâche fastidieuse et consommatrice de temps où une collaboration entre cogniticiens et experts du domaine est nécessaire pour comprendre si les corrections à effectuer portent sur la formalisation (erreur syntaxique) ou sur le modèle (erreur de sens). Les outils et méthodologies actuels de conception d'ontologies proposent des services de débogage pour aider à la réalisation de cette tâche. Cependant, dans des cas complexes, ces services sont loin de fournir l'assistance adéquate aux concepteurs d'ontologie : manque d'efficacité, manque d'explications sur les causes de l'insatisfiabilité d'une classe, manque de proposition de correction. Nous prétendons qu'il est possible de fournir une assistance supplémentaire aux concepteurs en utilisant une stratégie de débogage basée sur l'identification d'anti-patrons. Cette stratégie peut être associée à des outils existants pour faciliter le débogage.

Mots-clés : ontologies OWL-DL, débogage, conception d'ontologie.

1 Introduction

La construction d'une ontologie OWL-DL est une tâche difficile car son implémentation peut amener à la détection de contradiction dans le modèle sous jacent. Une des tâches des raisonneurs comme Fact++ ou Pellet est de valider la cohérence du modèle associé à l'ontologie. Un raisonneur peut invalider de différente manière une ontologie (Stuckenschmidt, 2008):

1. Une ontologie peut être détectée comme inconsistante : c'est-à-dire qu'il n'existe aucune interprétation possible de l'ontologie qui soit un modèle pour cette l'ontologie.
2. Une ontologie peut être détectée comme incohérente : c'est-à-dire qu'il existe une interprétation possible de cette ontologie qui soit un modèle, mais que ce modèle sous-jacent associe à certaines classes un ensemble d'instances qui sera toujours vide. C'est-à-dire qu'aucun objet ne pourra

jamais être une instance de cette classe. Ce type de classes est appelée classe insatisfiable.

Nos travaux s'intéressent à la correction de l'incohérence d'ontologies OWL-DL, c'est-à-dire à la résolution des contradictions qui provoquent l'insatisfiabilité d'une classe. Plusieurs outils de débogage existent : OWLDebugger (Horridge et al., 2008), SWOOP (Kalyanpur et al., 2006), RepairTab (Lam et al., 2008). Leur but principal est d'isoler le plus petit ensemble d'axiomes menant à l'insatisfiabilité d'une classe (Minimal Unsatisfiability Preserving Subset : MUPS). Les outils comme RepairTab (Lam et al., 2008) proposent aussi des solutions pour résoudre les contradictions identifiées avec une estimation de leurs conséquences sur les autres classes. Néanmoins les solutions sont toujours limitées à deux propositions : suppression d'une partie des axiomes existants ou remplacement d'une classe par une de ses superclasses. De plus, les ontologies utilisées dans leurs expérimentations ont été écrites par des experts en Logiques de Description (LDs) dans le but d'identifier des conflits particuliers.

Au contraire, nos travaux portent sur le débogage d'ontologies réelles qui ont été développées par des experts du domaine pas forcément très familiers avec les LDs. Par conséquent, ils peuvent employer improprement des constructeurs LDs ou comprendre imparfaitement la sémantique des expressions OWL-DL. Pour illustrer nos propos, nous utiliserons des exemples issus d'une ontologie OWL-DL de taille moyenne, développée par des experts en hydrologie, intitulée HydrOntology (Vilches-Blázquez et al., 2007). Les outils testés (Horridge et al., 2008), (Kalyanpur et al., 2006) lors du débogage n'étaient pas adaptés. Les informations fournies sur l'insatisfiabilité des classes n'étaient pas compréhensibles par les experts en hydrologie et les outils mettaient plusieurs heures pour calculer les MUPS, confirmant les résultats décrits dans (Stuckenschmidt, 2008). De plus, nous avons découvert que les experts du domaine lors du débogage de leur ontologie modifiaient d'une manière aléatoire les classes insatisfiables allant même jusqu'à changer la signification de leur modèle au lieu de corriger des erreurs dans leur formalisation.

Notre expérience dans le débogage d'ontologies et notre étude du comportement des experts du domaine, nous a permis d'identifier des patrons de conception utilisés dans la conception d'ontologies OWL-DL. Nous avons de plus cherché à définir des solutions à ces patrons. Ainsi nous proposons une stratégie de débogage d'ontologie OWL-DL basé sur ces patrons et les solutions associées pour résoudre les incohérences. Cette stratégie est associée à des outils existants de débogage et de détection de patrons. La dernière partie de nos travaux présente les résultats sur la détection des anti-patrons utilisant trois approches différentes.

2 Anti-patrons ontologiques

En génie logiciel, un patron de conception est défini comme une solution standard à un problème récurrent d'architecture et de conception logicielle. Issus de l'expérience, les patrons de conception représentent les meilleures pratiques de la conception logicielle. Au contraire, les anti-patrons sont définis comme des mauvais

choix de modélisation provoquant des erreurs, des comportements inattendus ou une lenteur à l'exécution (Koenig, 1998).

En ingénierie de connaissances, les patrons de conception ontologiques sont utilisés pour référencer des solutions de modélisation en réponse à un problème de conception (Clark et al., 2000; Presutti et al., 2008). Les anti-patrons sont définis de la même manière qu'en génie logiciel.

Au contraire des patrons de conception ontologique, les travaux sur les anti-patrons de conception sont beaucoup moins développés. La méthode ONTOCLEAN (Guarino & Welty, 2002) définit un ensemble de méta-propriétés sur les classes et les patrons associés pour vérifier et corriger les relations de subsomption entre classes. (Svab-Zamazal & Svátek, 2002) propose quatre patrons terminologiques sur les noms des classes pour détecter des erreurs possibles dans les relations de subsomption entre classes. (Guarino, 2009) présente quatre anti-patrons logiques portant uniquement sur les domaines et les codomaines des propriétés. (Rector et al., 2004) décrit des difficultés de compréhension et les erreurs couramment faites par les novices en LDs. Nos travaux s'apparentent à ceux de (Rector et al., 2004) et peuvent être vu comme un complément. L'apparition et la reconnaissance des anti-patrons de conception ontologique dépendent directement du niveau de compréhension des constructeurs de LDs par les utilisateurs. Plus l'utilisateur a du mal à saisir la sémantique du constructeur, plus il risque de mal l'utiliser. Notre expérience dans l'enseignement des LDs nous permet de savoir quels sont les constructeurs qui posent problème.

Cependant, aucun de ces travaux ne propose une classification des anti-patrons de conception, ni ne fournit des solutions pour corriger ces anti-patrons.

2.1 Classification de patrons de conception ontologique

Suite à la correction de l'ontologie HydrOntology et à notre expérience dans la conception d'ontologies et leur correction, nous avons identifié un ensemble d'anti-patrons qui sont utilisés généralement par des experts du domaine lors de leurs formalisations en avec OWL-DL. Certains de ces patrons ont pour conséquence la création de classes insatisfiables ou d'erreurs de modélisation. Comme mentionné auparavant, tous ces anti-patrons viennent d'une mauvaise compréhension des expressions en LDs par des concepteurs d'ontologie. Ainsi, ce sont tous des Anti-Patrons Logiques (LAP) : ils sont indépendants du domaine, mais spécifiques à l'expressivité du formalisme logique utilisé pour la modélisation. Nous les avons classés en trois catégories :

- Anti-Patrons Logiques Détectables (Detectable Logical Anti-Patterns : DLAP). Ils contiennent des conflits que les raisonneurs LDs détectent.
- Anti-Patrons Logiques Cognitifs (Cognitive Logical Anti-Patterns : CLAP). Ce sont des erreurs de modélisation qui peuvent être dues à une mauvaise interprétation des axiomes LDs.
- Conseils (Guidelines : G). Ils représentent des axiomes complexes utilisés dans la définition d'un élément de l'ontologie qui sont corrects d'un point de vue

logique et cognitif, mais que le concepteur aurait pu écrire d'une manière plus simple ou avec plus de précision.

Il est important de noter que les DLAPs produisent des classes insatisfiables qui sont automatiquement détectées par des raisonneurs LDs comme FACT++ ou Pellet. Ces raisonneurs ont l'inconvénient de ne pas fournir suffisamment d'information à l'utilisateur pour corriger les classes insatisfiables. Au contraire les CLAPs et les Gs, ne génèrent pas de classes insatisfiables mais il peut arriver qu'une combinaison de ces anti-patrons provoque l'insatisfiabilité d'une classe.

2.2 Bibliothèque d'anti-patrons de conception ontologiques

Dans cette section nous décrirons uniquement six des anti-patrons que nous avons identifiés lors du débogage de HydrOntology. Pour un complément d'information, nous invitons le lecteur à se reporter au catalogue des anti-patrons présenté dans (Roussey et al., 2009). Ces six anti-patrons sont ceux que nous jugeons les plus facilement utilisable par des non experts en LDs lors du débogage des ontologies. Des occurrences de ces anti-patrons sont apparues plusieurs fois dans le débogage de HydrOntology et notre expert du domaine après une explication a compris le problème. Dans la suite de cette section, ces anti-patrons sont présentés par ordre de difficulté de compréhension. Leur description contient :

- un nom et un acronyme,
- son type pris parmi les trois catégories DLAP, CLAP et G,
- sa formule en LDs,
- une brève explication des raisons pour lesquelles cet anti-patron peut apparaitre,
- un exemple en anglais issu de HydrOntology,
- une recommandation pour la correction de cet anti-patron. Pour chaque anti-patron il existe plusieurs solutions possibles. La recommandation que nous proposons est la solution qui a été validée par notre expert du domaine lors du débogage d'HydrOntology. Ainsi les cogniticiens experts en DL peuvent voir la différence qui existe entre ce que voulait dire l'expert du domaine et ce qu'il a écrit.

2.2.1 Anti-patron CLAP: SynonymOrEquivalence (SOE)$\boldsymbol{C1 \equiv C2}$

Le concepteur d'ontologie souhaite exprimer que deux classes C1 et C2 sont identiques. Dans une même ontologie cette relation d'équivalence n'est pas utile, elle est plutôt utilisée quand on veut faire correspondre deux classes issues d'ontologies différentes. En réalité ce que le concepteur veut exprimer est une relation de synonymie entre deux noms (labels) C1 ou C2 d'une même classe. Habituellement une des deux classes n'est pas utilisée dans les autres axiomes de l'ontologie.

$$Subterranean\_Watercourse \equiv Subterranean\_River$$

Fig. 1 –Exemple d'anti-patron SOE dans HydrOntology

Supposons que C2 est la classe la moins utilisée dans l'ontologie, la correction de SOE consiste à ajouter tous les commentaires et les labels de C2 dans C1. Ensuite il faut dans chaque axiome utilisant C2, le remplacer par C1, puis supprimer C2.

$$\mathrm{C1} \equiv \mathrm{C2} \Rightarrow C1.[rdfs{:}\,label|comment] = C2.[rdfs{:}\,label|comment],\ delete(C2)$$

2.2.2 Anti-patron DLAP: EquivalenceIsDifference (EID)

$$C1 \equiv C2; Disj(C1, C2)$$

Ce patron apparait uniquement chez les concepteurs d'ontologie débutant dans la formalisation en LDs. Le concepteur souhaite formaliser que les instances de C1 ressemblent aux instances de C2 avec une différence par exemple une information supplémentaire. Après une courte formation, les concepteurs découvrent qu'ils souhaitent exprimer une relation de subsomption entre C1 et C2 : C1⊑C2

$$Cascade \equiv Waterfall; Disj(\mathrm{Cascade}, Waterfall)$$

Fig. 2 –Exemple d'anti-patron EID dans HydrOntology

Pour résoudre cet anti-patron, il faut demander au concepteur s'il souhaite exprimer : une relation de synonymie entre deux noms d'une même classe ou une relation de subsomption entre deux classes. Selon sa réponse, l'équivalence devrait être transformée en relation de subsomption ou supprimer une des deux classes en suivant les recommandations de l'anti-patron SOE.

$$C1 \equiv C2; Disj(C1, C2) \Rightarrow C1 \sqsubseteq C2\ or\ C2\ is\ a\ label\ of\ C1$$

2.2.3 Anti-patron DLAP: AndIsOr (AIO)

$$\boldsymbol{C1} \sqsubseteq \exists \boldsymbol{R}.(\boldsymbol{C2} \sqcap \boldsymbol{C3});\ \boldsymbol{Disj}1\ (\boldsymbol{C2}, \boldsymbol{C3})$$

Cet anti-patron est une erreur de modélisation courante due à l'ambigüité des conjonctions de coordination « et » et « ou »qui ne correspondent pas forcément à la conjonction et à la disjonction logiques (Rector et al., 2004). Par exemple la phrase «j'aime les gâteaux avec du chocolat et des amandes» est ambiguë. Que contient réellement le gâteau ?

- du chocolat plus des amandes?

$$Cake \sqsubseteq \exists contain.Chocolate \sqcap \exists contain.Almond$$

- du chocolat aux amandes ? $Cake \sqsubseteq \exists contain.(Chocolate \sqcap Almond)$
- du chocolat ou des amandes? $Cake \sqsubseteq \exists contain.(Chocolate \sqcup Almond)$

$$Pipe \sqsubseteq \exists\ communicate.(Lagoon \sqcap Sea \sqcap Salt\_Marsh)$$

[1] Il est à noter que le concepteur d'ontologie n'a pas forcément formulé que C2 et C3 sont disjoints, mais que ces deux concepts sont déterminés comme disjoints par un raisonneur. Nous employons cette notation comme raccourci pour $C2 \sqcap C3 \sqsubseteq \bot$

Fig. 3 –Exemple d'anti-patron AIO dans HydrOntology

Pour résoudre cet anti-patron, nous proposons de remplacer la conjonction logique par la disjonction logique, ou par la conjonction de deux restrictions existentielles

$$C1 \sqsubseteq \exists R.(C2 \sqcap C3);\ Disj(C2, C3) \Rightarrow\ \#1\ C1 \sqsubseteq \exists R.(C2 \sqcup C3); Disj(C2, C3)$$

$$\#2\ C1 \sqsubseteq \exists R.C2\ and\ \exists R.C3;\ Disj(C2, C3)$$

2.2.4 Anti-patron G: DisjointnessOfComplement (DOC) $C1 \equiv not\ C2$

Le concepteur d'ontologie veut *exprimer que* C1 et C2 ne partagent aucune instance commune. Pour se faire, il serait plus approprié de définir que C1 et C2 sont disjoints au lieu de définir que C1 est le complément de C2. De plus il est difficile de définir le complément d'une classe au cours de la construction d'une ontologie car le nombre de classes n'est pas encore figé. Par exemple, le concepteur pourra avoir besoin de créer une nouvelle classe C3, disjointe de C1 et C2.

$$Salt\_Lagoon \equiv not\ Fresh\_Water$$

Fig. 4 –Exemple d'anti-patron DOC dans HydrOntology

La solution proposée est : $C1 \equiv not\ C2\ \Rightarrow Disj(C1, C2)$

2.2.5 Anti-patron DLAP: OnlynessIsLoneliness (OIL)

$$C1 \sqsubseteq \forall R.C2;\ C1 \sqsubseteq \forall R.C3;\ Disj(C2, C3)$$ [2]

Le concepteur d'ontologie a créé une restriction universelle pour indiquer que les instances de C1 peuvent être liées par la propriété R uniquement aux instances de C2. Au cours du développement de l'ontologie, une nouvelle restriction universelle est ajoutée indiquant que les instances de C1 peuvent être liées par R uniquement aux instances de C3, C2 et C3 étant disjoints. Généralement, le concepteur a oublié au cours du développement l'existence du premier axiome dans la classe C1 ou dans l'une de ses classes parentes.

$$Transitional\_Water \sqsubseteq \forall is\_nearby.Sea\_Water\ \sqcap \forall is\_nearby.River\_Mouth \sqcap = 1 is\_nearby.\mathrm{T}$$

Fig. 5 –Exemple d'anti-patron OIL dans HydrOntology

Pour corriger cet anti-patron, nous proposons à l'expert du domaine de fusionner les deux restrictions universelles en une avec une disjonction entre C2 et C3.

$$C1 \sqsubseteq \forall R.C2;\ C1 \sqsubseteq \forall R.C3; Disj(C2, C3) \Rightarrow\ C1 \sqsubseteq \forall R.(C2 \sqcup C3); Disj(C2, C3)$$

[2] Pour être détectable, la propriété R doit être instanciée au moins une fois sur une instance de la classe C1, en spécifiant soit une restriction de cardinalité, soit une restriction existentielle.

2.2.6 Anti-patrons DLAP : UniversalExistence (UE)

$$C1 \sqsubseteq \exists R.C2;\ C1 \sqsubseteq \forall R.C3;\ Disj(C2, C3)$$

Le concepteur d'ontologie ajoute une restriction existentielle (universelle) à une classe en oubliant qu'il existe déjà une restriction universelle (existentielle) dans cette classe ou l'une de ses classes parentes.

$$Inlets \sqsubseteq Sea\_Waters\_Canal;\ Inlets \sqsubseteq \exists communicate.Rivers;$$

$$Sea\_Waters\_Canal \sqsubseteq\ \forall communicate.Sea\_Water$$

Fig. 6 –Exemple d'anti-patron UE dans HydrOntology

Cet anti-patron est difficile à corriger parce que les concepteurs d'ontologie ne comprennent pas clairement la différence entre une restriction existentielle et une restriction universelle. La correction que nous proposons à pour but de résoudre l'insatisfiabilité d'une classe, mais elle devra être débattue avec le concepteur.

$$C1 \sqsubseteq \exists R.C2;\ C1 \sqsubseteq \forall R.C3;\ Disj(C2, C3)$$

$$\Rightarrow C1 \sqsubseteq \exists R.C2; C1 \sqsubseteq \forall R.(C2 \sqcup C3);\ Disj(C2, C3)$$

3 Stratégie de débogage d'ontologies

Comme introduit précédemment, des outils de débogage d'ontologies OWL-DL sont déjà disponibles (Horridge et al., 2008; Kalyanpur et al., 2006; Wang et al., 2005). Ils permettent d'identifier : les classes insatisfiables, les classes racines provoquant l'insatisfiabilité de plusieurs classes, les axiomes et les restrictions superflus, dans certains cas ils sont aussi capables d'expliquer l'insatisfiabilité d'une classe avec différents niveaux de détail. Ces informations permettent ainsi de guider efficacement le processus de débogage. Cependant, elles sont centrées sur les implications logiques et ne s'intéressent pas à la dimension ingénierie de la conception d'ontologies. De plus les concepteurs d'ontologie ont parfois des difficultés à comprendre ces implications logiques. D'autres travaux s'orientent sur la gestion des évolutions des ontologies en proposant des outils de transformation d'ontologies à base de patrons comme OPPL (Iannone et al., 2009) ou basés sur le langage SPARQL comme l'outil de détection d'anti-patron du projet PatOMat (Svab-Zamazal et al., 2009).

A notre connaissance, il n'existe pas à l'heure actuelle de stratégie précisant les étapes à suivre dans le processus de correction d'ontologies. C'est pourquoi, nous proposons une stratégie de débogage permettant aux concepteurs d'ontologie de détecter leurs erreurs et d'avoir accès à un ensemble de recommandations pour les corriger.

La figure 7 présente le cycle de vie de correction d'ontologies, en identifiant les rôles du cogniticien et de l'expert du domaine.

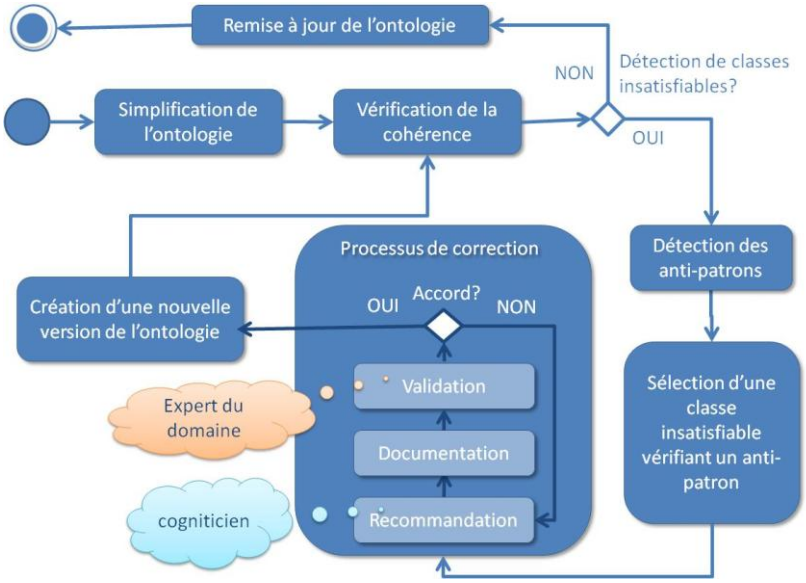

Fig. 7 –Stratégie de débogage d'ontologie.

3.1 Etape N°1 : Simplification de l'ontologie.

La première étape consiste à nettoyer l'ontologie pour extraire la taxonomie de classes et améliorer la performance des raisonneurs. De plus, un certain nombre d'erreurs peuvent parfois venir d'éléments secondaires non primordiaux pour la compréhension du modèle. Le nettoyage consiste à éliminer ces éléments secondaires :

- Suppression de toutes les instances de l'ontologie : ainsi une ontologie inconsistante deviendra une ontologie incohérente et l'utilisateur pourra travailler sur le modèle pour résoudre les classes insatisfiables.
- Suppression des domaines et des codomaines des propriétés : la suppression de ces informations peut éliminer un certain nombre de contradictions (voir les anti-patrons de (Guarino, 2009)). De plus il est difficile de figer le domaine ou codomaine d'une propriété lorsque la taxonomie de classes évolue.
- Suppression des propriétés transitives pour améliorer les performances du raisonneur: création pour chacune d'elle d'une propriété fille non transitive représentant les relations directes. Remplacement dans les axiomes de l'ontologie des propriétés transitives par leur propriété directe.

3.2 Etape N°2 : Vérification de la cohérence d'une ontologie.

Un raisonneur est utilisé pour déterminer l'ensemble des classes insatisfiables. Si l'ontologie est incohérente on passe à l'étape 3 sinon à l'étape 5.

3.3 Etape N°3 : Détection des anti-patrons et sélection d'une classe insatisfiable.

La troisième étape consiste à localiser les problèmes. Pour se faire deux solutions sont possibles :

1) Recherche automatique des anti-patrons en utilisant un outil de détection d'anti-patron comme OPPL (Iannone et al., 2009) ou celui du projet PatOMat (Svab-Zamazal et al., 2009) basé sur le language SPARQL.
2) Recherche manuelle des anti-patrons dans un ensemble d'axiomes contenant une contradiction (MUPS). Cet ensemble d'axiomes peut être calculé par un des outils de débogage d'ontologie (Wang et al., 2005).

L'étude décrite dans (Corcho et al., 2009) a montré qu'il était difficile pour les utilisateurs non experts en LDs de reconnaître un des anti-patrons dans les axiomes contenant un conflit. En effet nos anti-patrons sont des éléments unitaires qui peuvent être cachés dans des axiomes plus complexes. De plus il existe différentes manières d'écrire nos anti-patrons. Par exemple, un anti-patron peut être découpé dans plusieurs axiomes ou une de ses composantes peut être déduite par le raisonneur. Pour toute ces raisons, nous préconisons aux concepteurs non experts en LDs de travailler plutôt avec des outils de recherche automatique de patrons.

A la fin de cette étape l'utilisateur doit choisir une classe insatisfiable à corriger. Nous recommandons de commencer le débogage par les classes les plus hautes dans la taxonomie, car il y a plus de chance que leur correction entraine la correction d'autres classes, en particulier leurs classes filles. Si l'utilisateur a accès à un outil de débogage identifiant les classes racines d'incohérence, il faut choisir l'une de ces classes.

3.4 Etape N°4 : Correction d'une classe et création d'une nouvelle version de l'ontologie.

La correction d'une classe est dirigée par l'anti-patron découvert dans les axiomes définissant cette classe. Notre bibliothèque d'anti-patrons associe à chacun d'eux un ensemble de recommandations de correction. Le cogniticien propose à l'expert du domaine la recommandation qu'il juge la plus adéquate, en lui expliquant l'impact de cette modification sur le modèle. En effet une correction peut changer le sens du modèle. Ainsi, la correction d'une classe est une négociation entre le cogniticien et l'expert du domaine : c'est un processus itératif entre les propositions du cogniticien et les validations de l'expert du domaine. Pour se remémorer les étapes de la négociation, nous préconisons de toujours documenter ce processus de correction.

Pour documenter le processus de débogage, nous recommandons de sauvegarder chaque changement dans une nouvelle version de l'ontologie. Ainsi il sera plus facile de revenir à un état particulier de l'ontologie si jamais une des corrections est remise en cause.

3.5 Etape N°5 : Remise à niveau de l'ontologie.

Une fois l'ontologie corrigée, les éléments supprimés dans l'étape N°1 sont recréés dans l'ontologie.

4 Détection des anti-patrons

Nous avons utilisé nos anti-patrons pour déboguer trois ontologies incohérentes :

- HydrOntology : une ontologie sur l'hydrologie (Vilches-Blázquez et al., 2007) de 159 classes dont 114 sont détectées comme insatisfiables par Fact++.
- Tambis-full : une ontologie sur la biologie (Stevens et al., 2000) de 395 classes dont 144 classes sont détectées insatisfiables par Fact++.
- Ontologia-Forestal : une ontologie sur la gestion des domaines forestiers contenant 93 classes dont 62 sont détectées insatisfiables par Fact++.

Pour détecter nos anti-patrons trois expérimentations ont été menées :

1. Une détection manuelle dans l'ensemble des axiomes provoquant l'insatisfiabilité d'une classe au cours de la correction de l'ontologie. Nous avons utilisé Protégé 4 avec le raisonneur Fact++ et le plugin Explanation Workbench (Wang et al., 2005) qui permet de trouver les classes racines et les MUPS.
2. Une détection automatique avec une requête SPARQL et l'outil de détection associé au projet PatOMat (Svab-Zamazal et al., 2009) sur l'ontologie initiale.
3. Une détection automatique avec une requête SPARQL et l'outil de détection associé au projet PatOMat sur une ontologie qui est la matérialisation de toutes les inférences déduites par le raisonneur Pellet.

Le tableau suivant présente nos résultats pour chacun des anti-patrons.

Table 1. Nombre d'anti-patrons détectés pour chacune des méthodes.

Ontologie	Type de détection	SOE	EID	AIO	DOC	OIL	UE
HydrOntology	Manuelle	6	5	4	3	4	4
	SPARQL seul	12	6	4	3	4	5
Tambis	Manuelle	1	2	0	0	0	0
	SPARQL+ Pellet	20594	20592	2	0	0	0
	SPARQL seul	4	0	11	0	0	4
Ontologia Forestal	Manuelle	0	0	1	0	0	0
	SPARQL+Pellet	3782	3782	1	2821	0	0
	SPARQL seul	0	0	1	0	0	0

La détection manuelle a trouvé très peu d'anti-patrons. Principalement parce que la recherche de ces patrons s'arrête dès que l'ontologie est cohérente, même si

certaines mauvaises constructions du modèle persistent (par exemple SOE) dans l'ontologie.

La détection automatique est dépendante du langage SPARQL utilisé. La traduction d'un des nos anti-patrons en requête SPARQL n'est pas automatique. En effet il existe plusieurs versions d'un même anti-patron dépendant à la fois du style d'écriture de l'auteur et de la longueur moyenne des axiomes de l'ontologie. Certains auteurs préfèrent combiner dans un seul axiome toutes les connaissances permettant de définir une classe (Par exemple C1 subClassOf C2 and R some C5 and R only C7). Et d'autres préfèrent plutôt ajouter plusieurs axiomes ne contenant qu'une seule classe (Par exemple C1 subClassOf C2 ; C1 subClassOf R some C5 ; C1 subClassOf R only C7). De plus, certains de nos anti-patrons sont composés de plusieurs composants (par exemple UE) et ces composants peuvent se trouver dans des positions différentes dans les axiomes de l'ontologie définissant la même classe. Ainsi il nous faut construire un ensemble de requêtes SPARQL pouvant correspondre aux différentes versions d'écriture d'un de nos anti-patrons. Par exemple pour UE nous avons créés 45 requêtes différentes.

La détection automatique avec raisonneur signifie que l'on ne travaille plus sur les axiomes écrits par le concepteur de l'ontologie mais sur l'ensemble des axiomes inférés par le raisonneur. Par exemple on note un net accroissement du nombre d'axiomes traduisant une relation de disjonction entre ceux écrits directement par le concepteur de l'ontologie et ceux déduits par le raisonneur. Ce qui explique le nombre important de EID et DOC trouvés dans Tambis et Ontologia-Forestal. Pour éviter ce problème nous avons enlevé dans les requêtes SPARQL utilisées sans raisonneur la contrainte de la relation de disjonction entre concepts pour les anti-patrons AIO, OIL et UE. Ce qui explique que maintenant on retrouve plus d'anti-patrons AIO sans Pellet qu'avec Pellet vu que les contraintes sont moins strictes.

En conclusion la traduction des anti-patrons en requêtes SPARQL à besoin d'être améliorée pour couvrir l'ensemble des versions d'écritures possibles d'un anti-patron. Tambis et Ontologia_Forestale ne contenaient pas d'occurrences de OIL et UE, donc nous devrons tester nos requêtes sur d'autres ontologies. De plus, une liste d'occurrences d'anti-patrons n'est pas suffisante pour déterminer la classe insatisfiable à corriger.

5 Conclusion et Perspective

Dans cet article, nous avons décrit une stratégie pour déboguer les ontologies OWL-DL qui utilise un ensemble de services existants : raisonneurs, outils de détection d'anti-patrons**,** outils de résolution d'incohérences. Notre stratégie améliore l'efficacité du processus de débogage en proposant un ensemble d'actions prédéfinies à exécuter par les concepteurs d'ontologies. Cette stratégie est une synthèse de nos expériences sur le développement des ontologies LDs et le résultat de notre analyse sur la manière dont les concepteurs d'ontologies (cogniticien et expert du domaine) corrigent leurs ontologies. Nous avons mené une première série d'expérimentations sur la détection de nos anti-patrons, nous permettant d'évaluer la faisabilité de notre

approche. Nos travaux futurs s'orienteront sur l'amélioration de la détection de nos anti-patrons et l'ordonnancement des classes à corriger.

Remerciement

Ces travaux sont partiellement soutenus par la Czech Science Foundation (projet PatOMat : grant P202/10/1825).

Références

CLARK, P., THOMPSON, J., & PORTER, B. (2000). *Knowledge Patterns.* the 7th International Conference Principles of Knowledge Representation and Reasoning, USA. 591-600

CORCHO, O., ROUSSEY, C., VILCHES-BLAZQUEZ, L. M., & PEREZ, I. (2009). *Pattern-based OWL Ontology Debugging Guidelines* Workshop on Ontology Patterns WOP, USA. CEUR-WS.org 516

GUARINO, N. (2009). Collection of anti-patterns and modeling errors Laboratory of Applied Ontology.

GUARINO, N., & WELTY, C. (2002). Evaluating ontological decisions with ontoclean. *Communications of the ACM, 45*(2), 61-65.

HORRIDGE, M., PARSIA, B., & SATTLER, U. (2008). *Laconic and Precise Justifications in OWL.* ISWC, , Germany. LNCS 5318. 323-338.

IANNONE, L., RECTOR, A., & STEVENS, R. (2009). *Embedding Knowledge Patterns into OWL.* ESWC, Greece. LNCS 5554. 218-232.

KALYANPUR, A., PARSIA, B., & CUENCA-GRAU, B. (2006). *Beyond Asserted Axioms: Fine-Grain Justifications for OWL-DL Entailments.* DL, UK. LNCS 4011: 170-184.

KOENIG, A. (1998). Patterns and Antipatterns. In L. Rising (Ed.), *The patterns handbook: techniques, strategies, and applications* (pp. 383-390). Cambridge: Cambridge University Press. 383 390.

LAM, J., SLEEMAN, D., PAN, J., & VASCONCELOS, W. (2008). A Fine-Grained Approach to Resolving Unsatisfiable Ontologies. In *Journal on Data Semantics X* (Vol. 4900/2008, pp. 62-95): Springer.

PRESUTTI, V., GANGEMI, A., DAVID, S., AGUADO, G., SUAREZ-FIGUEROA, M. C., MONTIEL, E., et al. (2008). *Neon Deliverable D2.5.1: A Library of Ontology Design Patterns*.

RECTOR, A., DRUMMOND, N., HORRIDGE, M., ROGERS, J., KNUBLAUCH, H., STEVENS, R., et al. (2004). *OWL pizzas: Practical experience of teaching OWL-DL: Common errors and common patterns.* EKAW, LNCS 3257, p 63-81

ROUSSEY, C., CORCHO, O., & VILCHES-BLÁZQUEZ, L. M. (2009). *A catalogue of OWL ontology antipatterns.*, K-CAP, USA. 205-206.

STEVENS, R., BAKER, P., BECHHOFER, S., NG, G., JACOBY, A., PATON, N. W., et al. (2000). TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. *Bioinformatics, 16*(2), 184-185.

STUCKENSCHMIDT, H. (2008). *Debugging OWL Ontologies-A Reality Check.* Paper presented at the 6th International Workshop on Evaluation of Ontology-based Tools and the Semantic Web Service Challenge, EON, Tenerife, Spain. CEUR-WS.org 359

SVAB-ZAMAZAL, O., SCHARFFE, F., SVATEK, V. (2009). *Preliminary Results of Logical Ontology Pattern Detection using SPARQL and Lexical Heuristics* Workshop on Ontology Patterns, WOP, Washington D.C., USA, CEUR-WS.org:516 p.

SVAB-ZAMAZAL, O., SVÁTEK, V. (2002). *Analysing Ontological Structures through Name Pattern Tracking.* EKAW, Acitrezza, Italy. LNCS 5268: 213-228

TSARKOV, D., & HORROCKS, I. (2006). *FaCT++ description logic reasoner: System description.* IJCAR, , USA. LNAI 4130: 292-297.

VILCHES-BLÁZQUEZ, L. M., BERNABÉ-POVEDA, M. A., SUÁREZ-FIGUEROA, M. C., GÓMEZ-PÉREZ, A., RODRIGUEZ-PASCUAL, A. F. (2007). Towntology & hydrOntology: Relationship between Urban and Hydrographic Features in the Geographic Information Domain. In *Ontologies for Urban Development* (Vol. 61, pp. 73-84): Springer.

WANG, H., HORRIDGE, M., RECTOR, A., DRUMMOND, N., & SEIDENBERG, J. (2005). *Debugging OWL-DL ontologies: A heuristic approach.* ISWC, Ireland. LNCS 3729: 745-757.